Objectives

- Review Servlets
- Deployment
- Configuration
- Sessions, Cookies

Servlets Review

- What application do we need to execute servlets?
  - Which web application server are we using?
  - How is it different from a web server?
    - Relatedly: Why was Lab 4 not in our public_html directory?
- What class do all web servlets extend?
- What methods do servlets need to override to handle GET and POST requests, respectively?
- How do servlets send an HTML document/response to the client?
- How do servlets get data from the client?
  - How does this relate to what is in the form?

Spring Term be like …

The Jerk

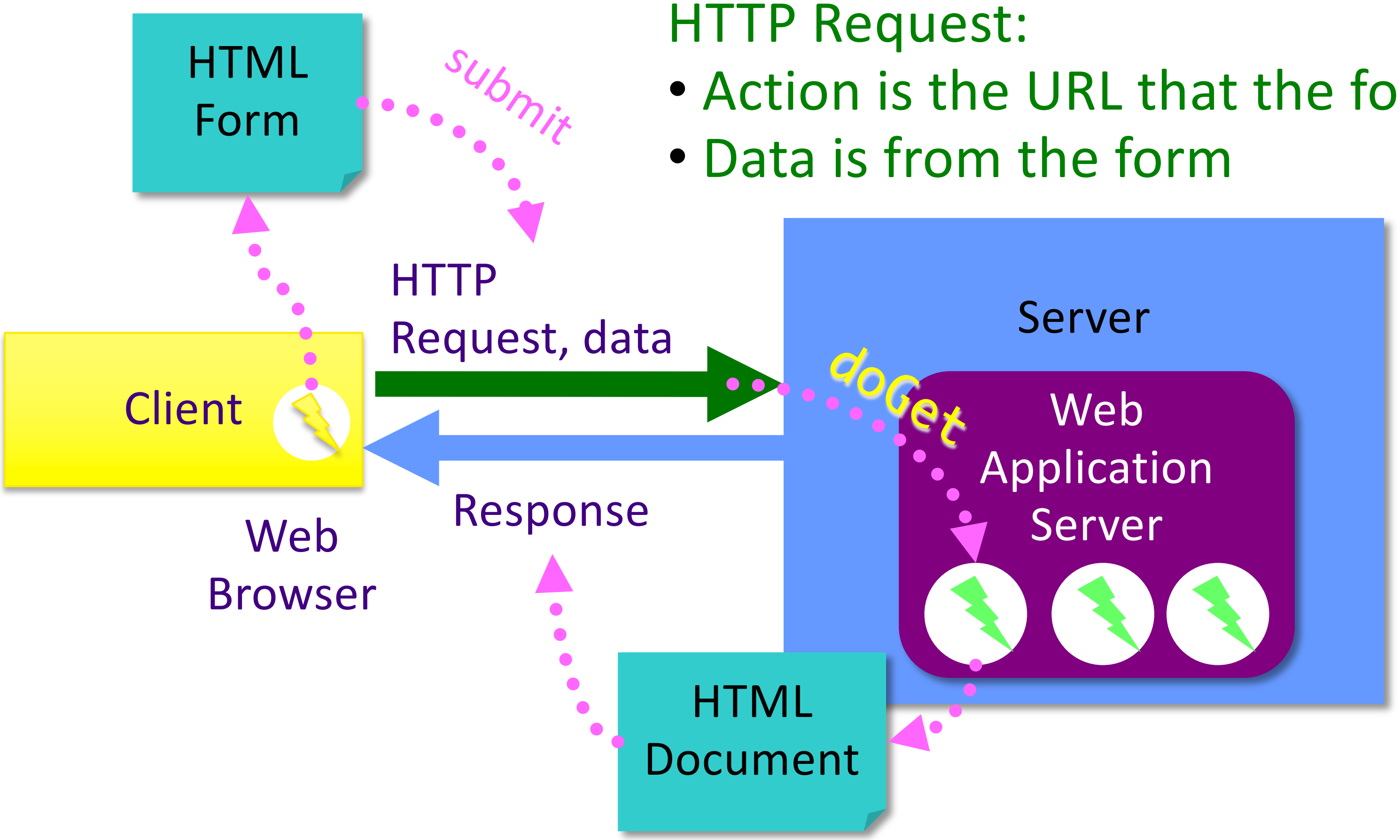
Servlets Review

- What is the structure of our dynamic web project in Eclipse?
  - What are the directories and what should they contain?
- When should we call
  `request.getParameter(String pname)` vs
  `request.getParameterValues(String pname)`
- Put it all together: how do you create a dynamic web page, i.e., a web page that processes a request from a form?
  - How do we connect things together? You have a bunch of different pieces, so how do they relate?
- What tricks did you learn to help you with debugging?

Example Servlet Flow

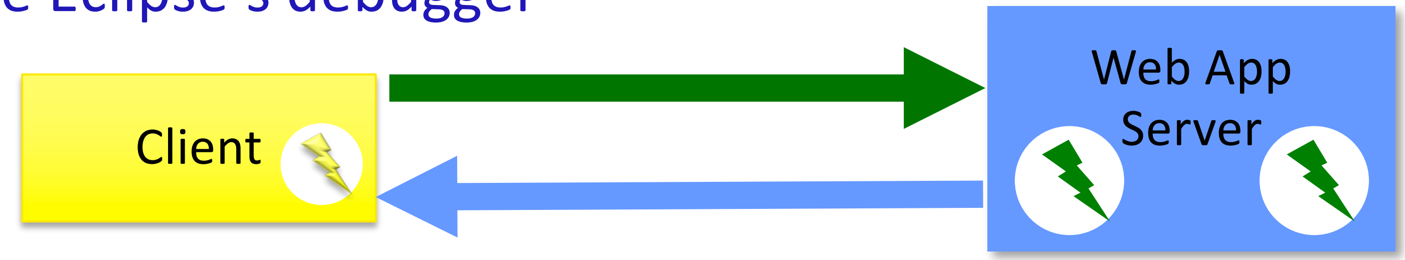
HTTP Request:

- Action is the URL that the form is sent to
- Data is from the form



Servlet Development Discussion

- Distributed applications are difficult to debug and test
  - Multiple components: Client code? Server code?
- Suggestions
  - Use Eclipse to help you find errors in HTML
  - Check response’s HTML source code
    - Shows you what was written to output
    - Location of error
  - Print statements: in the server’s log
  - Use Eclipse’s debugger



Eclipse Development Hints

- Safe bet: restart server whenever change to a servlet
  - Can modify Server’s configuration, under Publishing
- Edit web.xml if you make changes to servlet file names/packages
- Typical programming
  - Write a few lines of code/make small changes
  - Run, test
  - Repeat

More on Java-based Web Applications

- Structure
- Other classes
- Initialization, customization

Web App Directory Structure

- `projectname/`
    - HTML, CSS, and JSP files
- `projectname/WEB-INF`
    - Other resources, e.g., `web.xml`
- `projectname/WEB-INF/classes`
    - Servlet and utility (data structures, etc)
    - Why we put our servlets in `servlets` package
- `projectname/WEB-INF/lib`
    - Jar files that application depends on

Web App Directory Structure

- Different from Eclipse's code organization
- When Eclipse deploys the web application, it organizes it this way.

- `projectname/`
  - HTML, CSS, and JSP files
- `projectname/WEB-INF`
  - Other resources, e.g., `web.xml`
- `projectname/WEB-INF/classes`
  - Servlet and utility (data structures, etc)
  - Why we put our servlets in `servlets` package
- `projectname/WEB-INF/lib`
  - Jar files that application depends on

During lab, check out

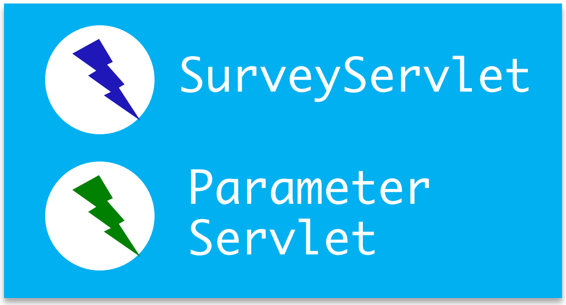
```
/path_to_your_eclipse_workspace/.metadata/
.plugins/org.eclipse.wst.server.core/tmp0/wtpwebapps/
```

`Servlet` Interface Methods

Recall: `HttpServlet` *implements* the Servlet interface

- `void init(ServletConfig config)`
  - Web app server calls once to initialize the servlet
  - Typically opening DB connection, files
- `ServletConfig getServletConfig()`
  - Returns a reference to a `ServletConfig`
- `void service(ServletRequest, ServletResponse)`
  - Called to respond to a client request
- `String getServletInfo()`
  - Returns a String that describes the servlet (name, version, etc.)
- `void destroy()`
  - Called by the server to terminate a servlet
  - Should close open files, close DB connections, etc.

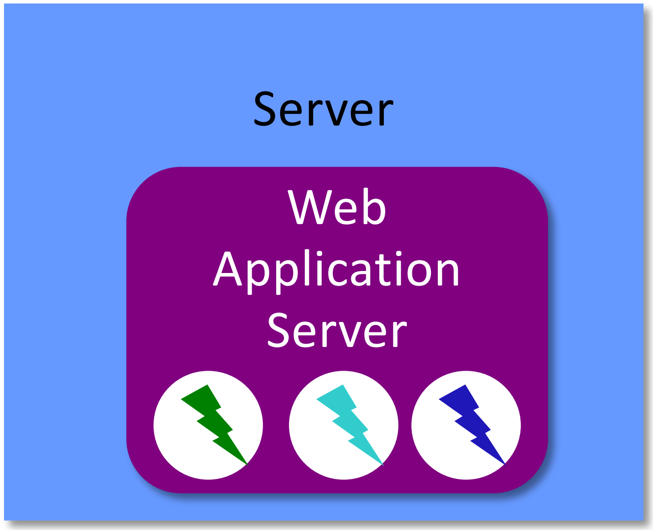
Servlet Life Cycle in Web Application Server (WAS)



Web Application Server

1. Web application server creates **one** instance of servlet
   - Calls `init` method of servlet created
2. As requests come in, WAS calls **`service`** method of appropriate servlet
   - In turn, servlet calls appropriate `doMethod`
3. When web application server shuts down, calls `destroy` method of each servlet

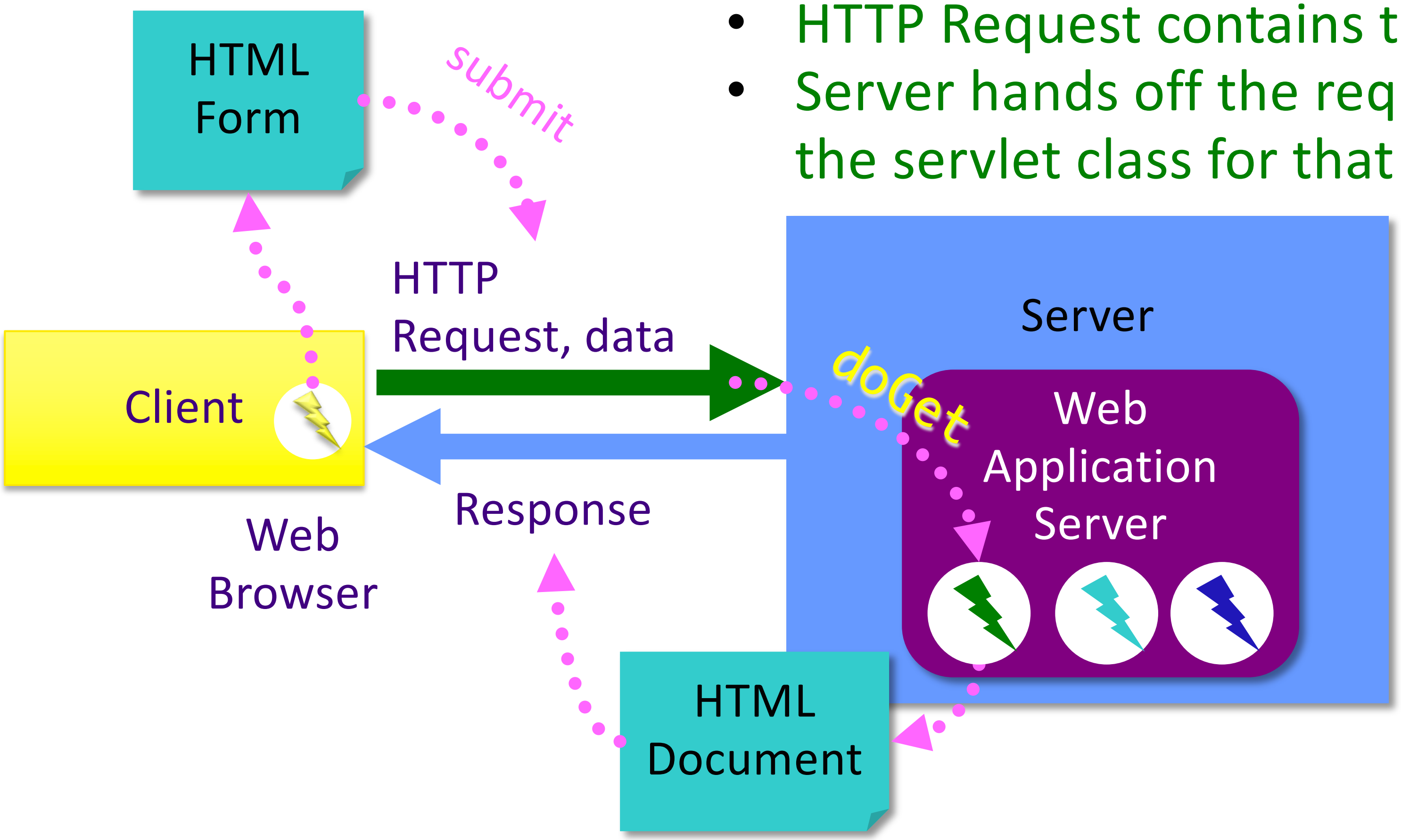
Web Application Server Flow



- Each of these is the one (and only) instance of *that* servlet class.
  - WAS calls `init` on each servlet class
- Each servlet has at least one URL mapping
  - URL → Servlet class

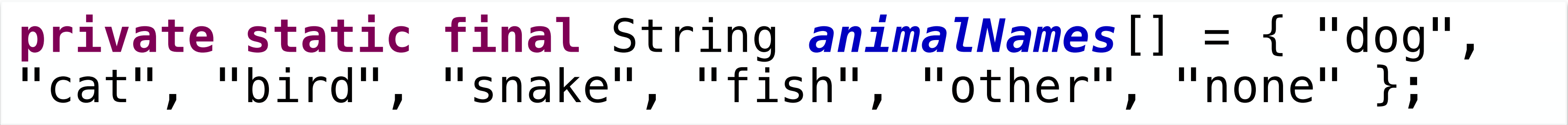
Example Servlet Flow

- HTTP Request contains the URL
- Server hands off the request to the servlet class for that URL



Java Review

- What does **static** mean?
- What does **final** mean?
- In SurveyServlet, why were the animal names static and final?
    - Did they need to be?



```
private static final String animalNames[] = { "dog",
"cat", "bird", "snake", "fish", "other", "none" };
```

Java Review

- Probably should be `final`
- `static` is not necessary in terms of there being only one object of each servlet type created
    - However, if other code needs to refer to those variables, it's easier to refer to them by the classname rather than the object name (how would you get a reference to that object?)

Lab 5: Refactoring `SurveyServlet`

- Currently: Inefficient implementation
  - Read, write survey data file every time request is executed
- In `init`
  - Automatically called by server on start up
  - Open file, read/initialize votes
- In `destroy`
  - Automatically called by server
  - Write file

Servlet Data

- `ServletConfig` – initialization and startup parameters for this servlet
  - Example methods:
    - `String getInitParameter(String name)`
    - `String getServletName()`

- `ServletContext` – servlet container information
  - Example methods:
    - `Object getAttribute(String name)`
    - `String getInitParameter(String name)`

Same method name, different context

ServletContext

- One `ServletContext` per web application per JVM
  - If you have both Lab4 and First running on Tomcat, they will each have their own ServletContext
- Share state among multiple clients within the web application
  - Allow multiple users to interact in, e.g., chat rooms, online meeting, reservation systems
- Info about servlet’s environment
  - E.g., server’s name
- `log()`: method to write to a log file
- Context attributes
  - `getAttribute, setAttribute, removeAttribute`

`web.xml` File

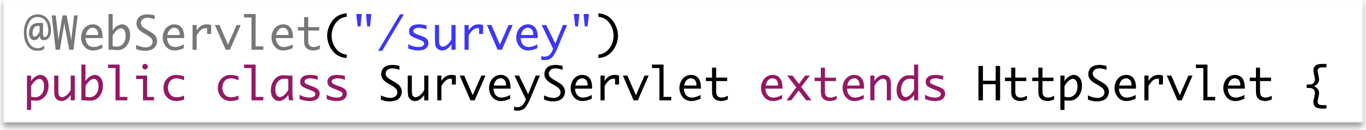
- Describes how to deploy the web application
- XML file
  - Used for data
  - Marked up with elements
  - Same rules as XHTML: close most recently opened tag, attributes in quotes

```
<tag attr="value">
   Content
</tag>
```

- DTD: Document Type Definition
  - Define elements that can be in a particular XML document
  - Includes specification of attributes, nesting

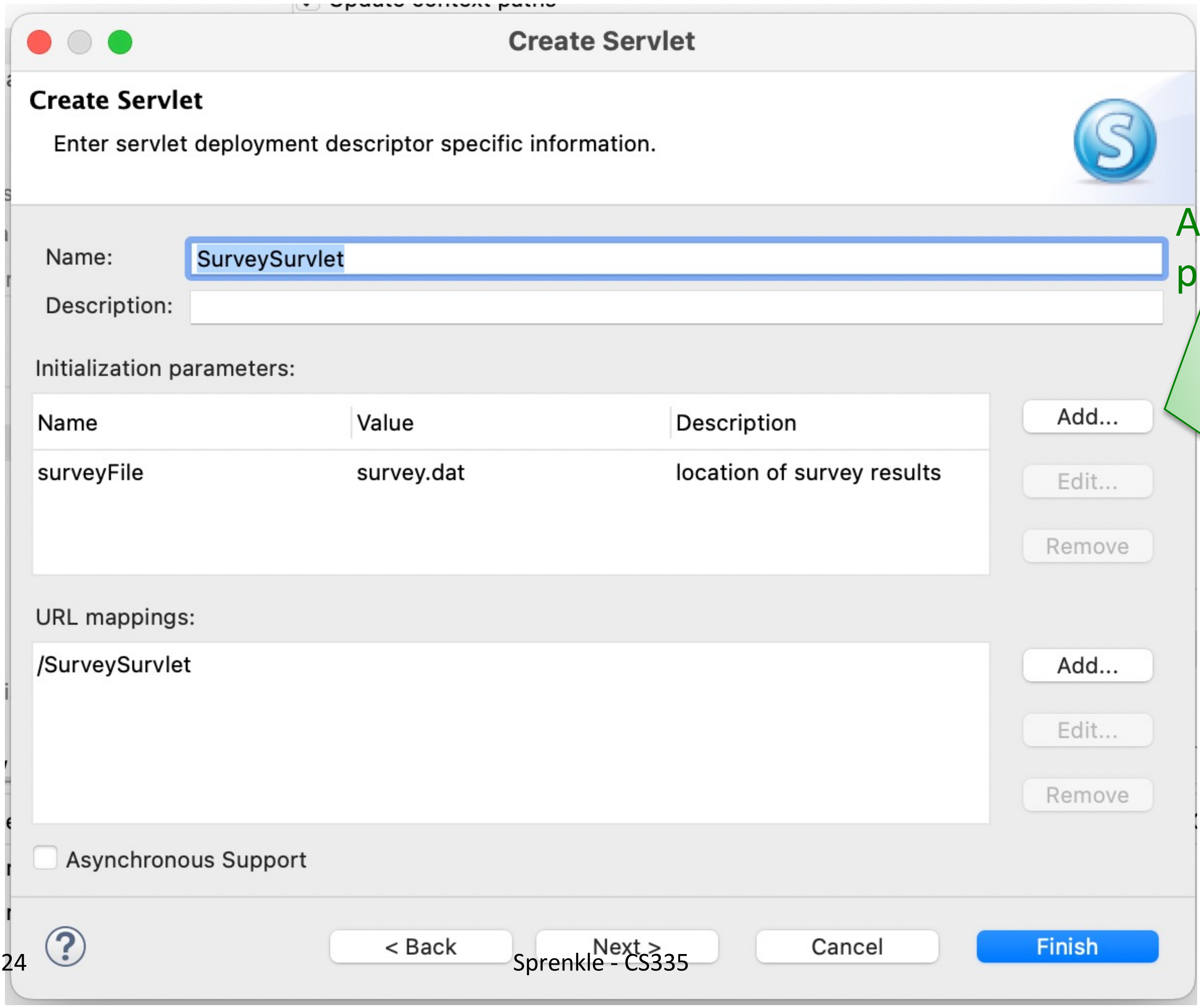
Annotations

- In Servlets 3.x, we can easily configure a web application using ***annotations***
  - Don’t need to directly update web.xml
  - Provide defaults, can be overridden in web.xml
- Example:



```
@WebServlet("/survey")
public class SurveyServlet extends HttpServlet {
```

  - Means the URL pattern “/survey” maps to this servlet (servlets.SurveyServlet)



Create Servlet

Enter servlet deployment descriptor specific information.

Name: SurveySurvlet

Description:

Initialization parameters:

Name	Value	Description
surveyFile	survey.dat	location of survey results

URL mappings:

/SurveySurvlet

□ Asynchronous Support

Add init parameters

Annotation Example with Init Parameters

Recall previous version:

```
@WebServlet("/survey")
public class SurveyServlet extends HttpServlet {
```

Extended version:

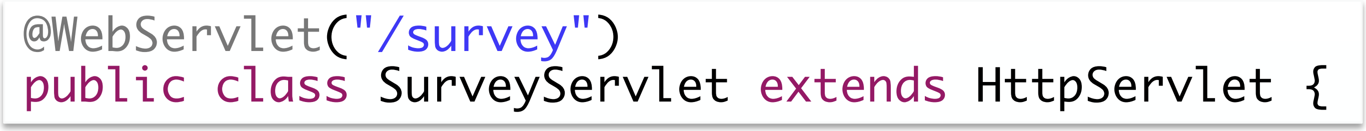
```
@WebServlet(
    name = "SurveyServlet",
    urlPatterns = { "/survey" },
    initParams = {
    @WebInitParam(name = "surveyFile",
           value = "survey.dat")
       })
public class SurveyServlet extends HttpServlet {
```

Must include the name if also have configuration for this servlet in web.xml (name must match servlet-name)

Why would we want init parameters?

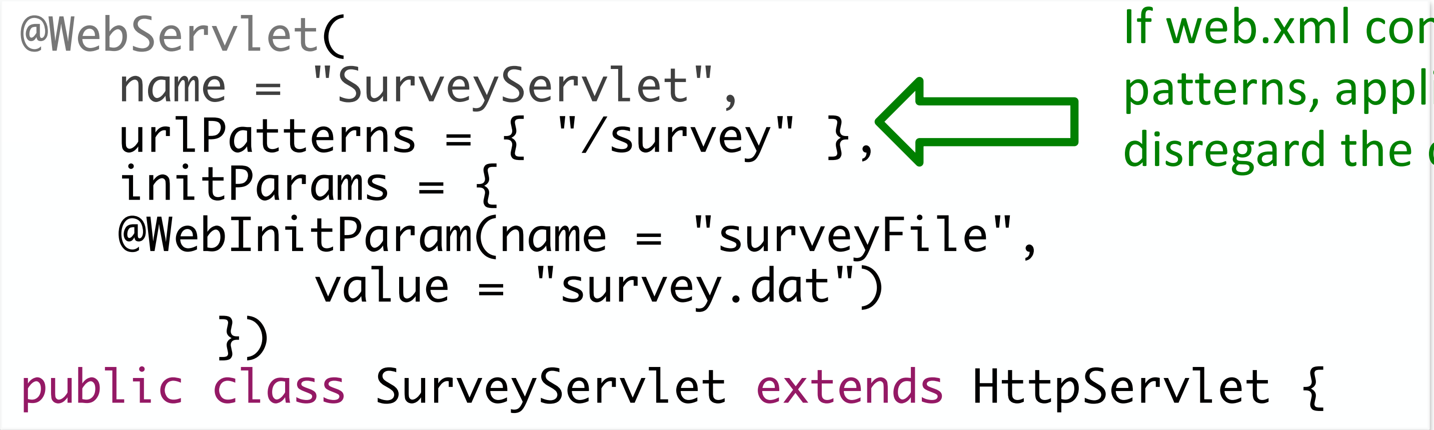
Annotation Example with Init Parameters

Recall previous version:



```
@WebServlet("/survey")
public class SurveyServlet extends HttpServlet {
```

Extended version:



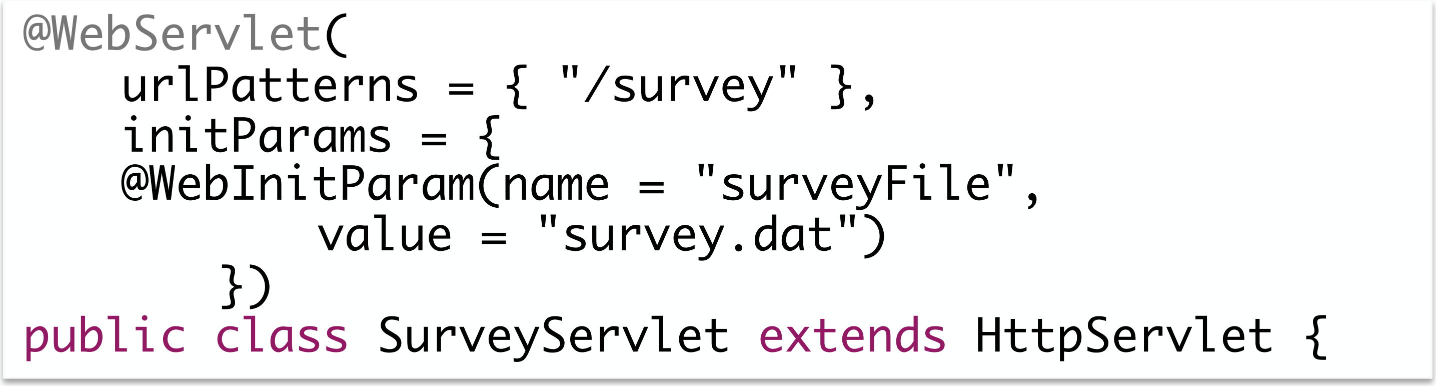
```
@WebServlet(
    name = "SurveyServlet",
    urlPatterns = { "/survey" },
    initParams = {
    @WebInitParam(name = "surveyFile",
            value = "survey.dat")
        })
public class SurveyServlet extends HttpServlet {
```

If web.xml contains other URL patterns, application will disregard the ones set here

Init Parameters Discussion

- Software should be *soft!*
- Want to be able to easily find/change parts of our software

Annotation Example with Init Parameters



```
@WebServlet(
    urlPatterns = { "/survey" },
    initParams = {
    @WebInitParam(name = "surveyFile",
           value = "survey.dat")
       })
public class SurveyServlet extends HttpServlet {
```

Annotation: Default values
Can override these in the web.xml

Why would we want to be able to
override these values in a separate (text) file?

Why override in web.xml?

- Can modify behavior of application ***without*** modifying the Java code and recompiling
  - May not have access to source code
- All configuration in one file
  - Don’t need to find which servlet it is in

web.xml File

- Top-level: <webapp>
- <servlet> element describes a servlet
- <servlet-mapping> element maps URLs to servlets
    - May want to have shorthands, aliases
    - Restrict users' direct access to servlets

web.xml File: Subelements of <servlet>

<servlet-name>	canonical name of the deployed servlet
<servlet-class>	fully qualified class name of the servlet
<init-param>	optional parameter containing a name-value pair that is passed to the servlet on initialization. Contains elements, <param-name> and <param-value>, which contain the name and value, respectively, to be passed to the servlet.

Example of Configuring web.xml

- Configure `SurveyServlet` to use a given file
- Add the following to web.xml file:

```
<init-param>
   <param-name>surveyFile</paramname>
   <param-value>survey.dat</param-value>
</init-param>
```

- Note that `<init-param>` is a child of `<servlet>`, which means your web.xml file would look like what?

Using Init Parameter

- Configure SurveyServlet to use a given file
  - Either in annotation or web.xml

- Modify init method to call HttpServlet's getInitParameter method

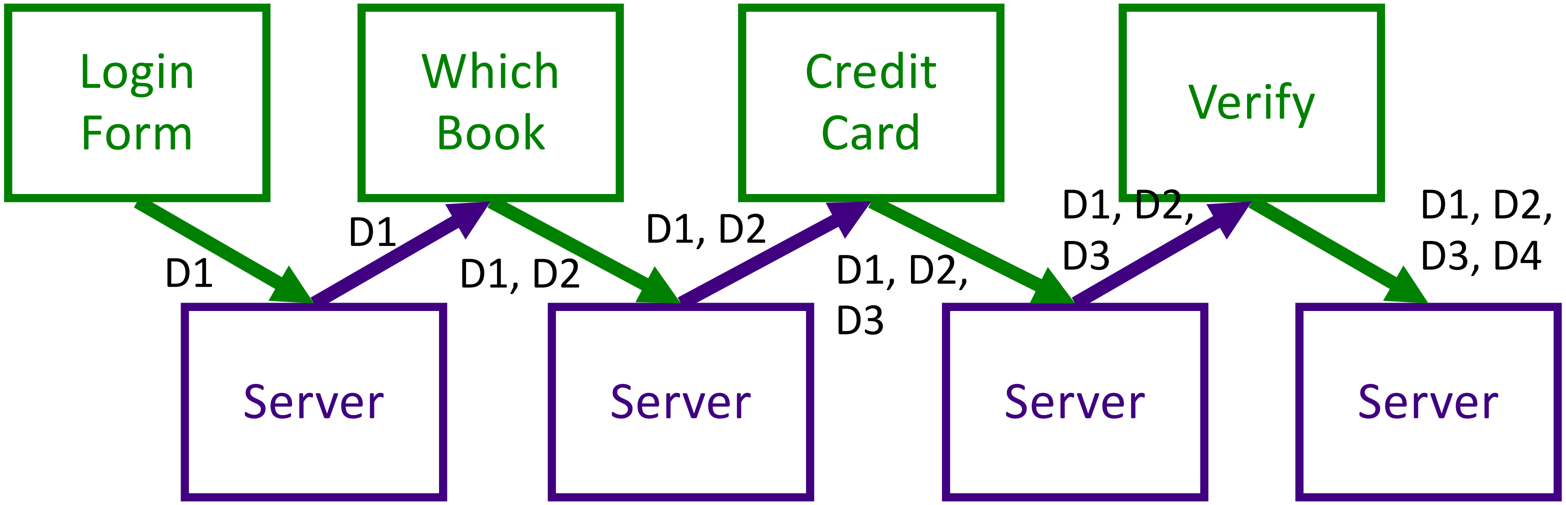
```
// calls HttpServlet method, i.e., this's method
filename = getInitParameter("surveyFile");

// create and open file …
```

MAINTAINING STATE ACROSS REQUESTS

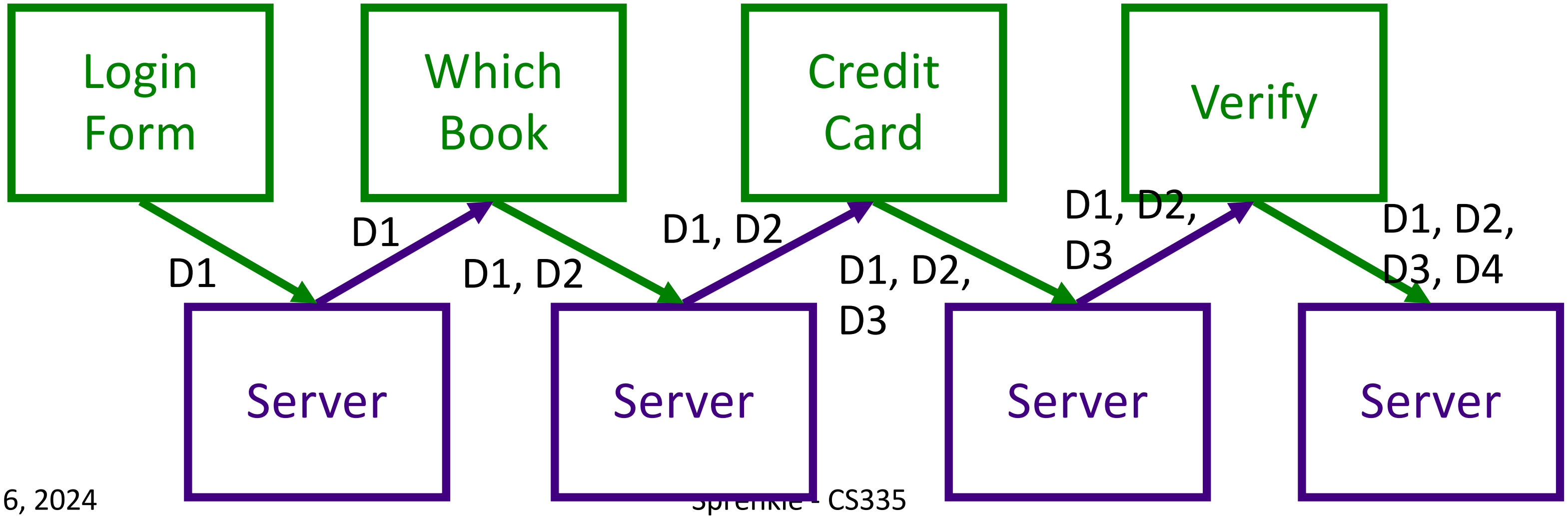
Maintaining State

- If you have multiple pages, how can you save or accumulate data?
  - Example scenario: buying a book

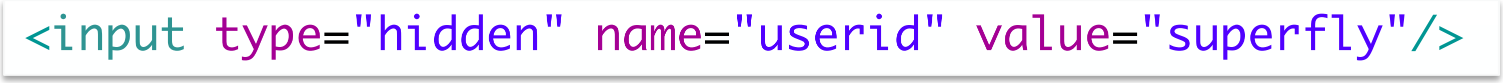


Maintaining State

- If you have multiple pages, how can you save or accumulate data?
  - Hidden fields (`type=hidden`)
  - Cookies
  - Sessions



Hidden Fields



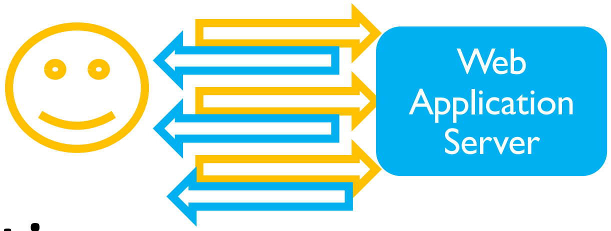
```
<input type="hidden" name="userid" value="superfly"/>
```

- Data is coming from client
- Users can see the hidden fields
  - View HTML Source
- Users can change the data

➡ Useful in limited situations

SESSION STATE

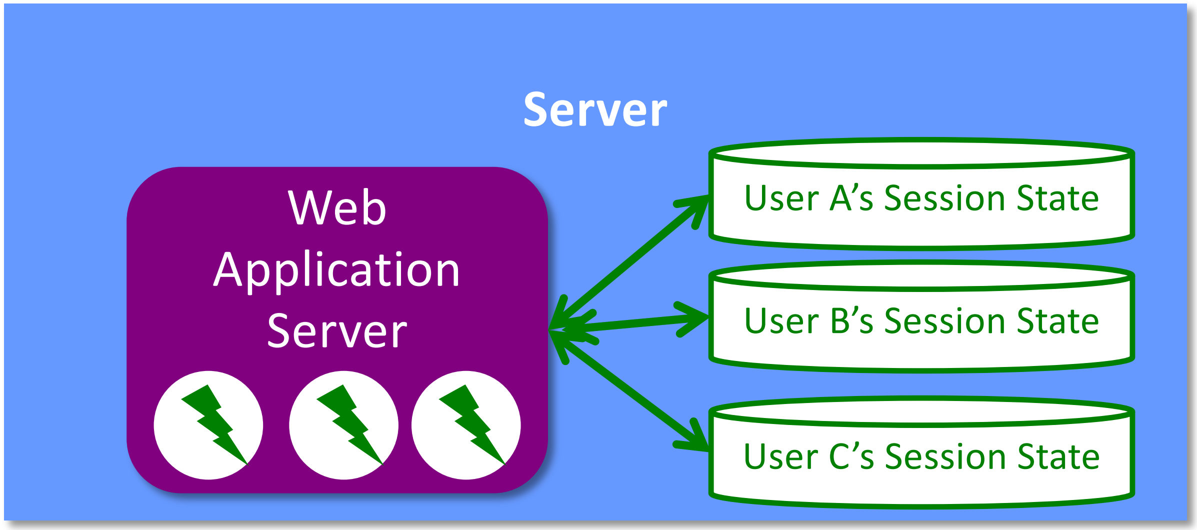
Session



- One user's visit to an application
- Can be made up of many requests
- Server maintains a session with a particular client
  - Can maintain ***state*** within that session
- Duration of a session:
  - If no requests from client for specified period of time (the timeout), user's session ends
  - Timeout: typically 30 minutes

Benefits of Using Session State

- Simpler for developer
- Reduces network traffic
  - Don’t need to keep passing data between client and server



Session State in Java

- `HttpSession` stores session data
- Data is known as ***session attributes***
  - Have names and values
- Store, access, and remove attributes:
  - Like a `HashMap`
  - `void setAttribute(String name, Object value)`
    - Values no longer need to be strings
    - Cookies and Parameters had to be strings
  - `Object getAttribute(String name)`
  - `void removeAttribute(String name)`

Example Session Variables

- User gives application data
- Application stores data in session variable

  name value

  -
    ```
    session.setAttribute("username", username);
    ```

- Application can use later in session, without user having to give information again

  -
    ```
    String username = (String)
                session.getAttribute("username");
    ```

- More examples:
  - Server computes information once, caches in session
  - Shopping carts

Getting a Session

- `HttpServletRequest`'s `getSession(boolean create)` method
  - Returns the current `HttpSession` object
  - Boolean parameter specifies if a new session should be created if one does not already exist

Other Useful Session Methods

- `setMaxInactiveInterval(), getCreationTime(), getLastAccessedTime()`
  - If want shorter than server's timeout

- `invalidate()`
  - Invalidates session, unbinds objects bound to it

Challenges/Tradeoffs to Using Session State?

Challenges/Tradeoffs to Using Session State?

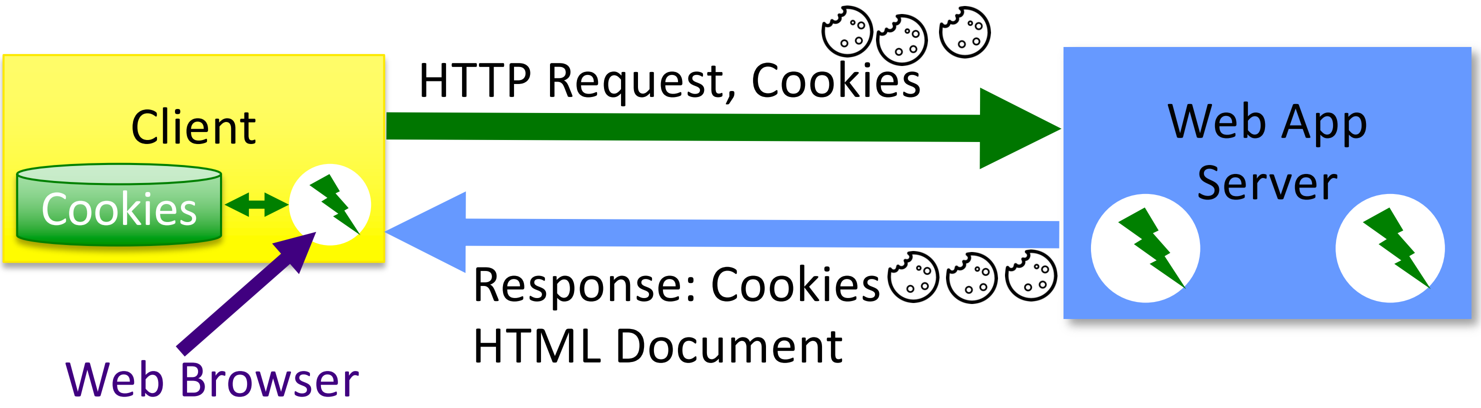
- Server needs to store the state for all users
  - That could be a lot of state and there are constraints on the server's memory
  - If memory gets tight, server can throw out session state
    - May save data to a data store before throwing it out
- For user, can be an annoyance if your session times out and you weren't done.

COOKIES

Cookies

- Cookies are initially sent from the webapp to the client to store application-specific information on the client
- Part of an HTTP header in response to a client
  - Every HTTP transaction includes HTTP headers
  - Not part of the HTML content
- Client includes cookies in HTTP headers in subsequent requests
  - Provides way to do behavior tracking

Process with Cookies



- Cookies
  - Associated with server name
  - Part of HTTP Headers
- Example: Amazon.com
  - Cookie stores your name, login information
  - Example: Not Sara?

Cookies in Java

- Cookies have a name and value
- Create a Cookie object using its constructor
  - Part of `jakarta.servlet.http.Cookie`
- Example: store a user's preferred language on the client
  - App only has to ask for this information once

```
String cookie_name = "pref_language";
String cookie_value = "English";
Cookie new_cookie = new Cookie(cookie_name, cookie_value);
```

Sending the Cookie to the Client

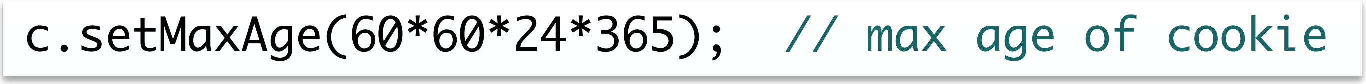
- HTTP header is sent first
- Cookie(s) must be added to the response object ***before*** you start writing to the client
- Call `addCookie`() on `HttpServletResponse` object before you call the `getWriter`() method
- Inside of **`doGet`** or **`doPost`** method:

```
Cookie c = new Cookie( "pref_language", "English" );
c.setMaxAge(60*60*24*365);  // max age of cookie
response.addCookie(c);
…
output = response.getWriter();
```

`HttpServletResponse` Method

- `void addCookie(Cookie)`
  - Add a Cookie to the header in the response to the client
  - The cookie will be stored on the client, depending on the max-life and if the client allows cookies

Cookies: Maximum Ages



```
c.setMaxAge(60*60*24*365);  // max age of cookie
```

- The maximum age of the cookie is how long the cookie can live on the client, in seconds
- When a cookie reaches its maximum age, client deletes it
- -1 means persists until browser exits

Retrieving Cookies

- Call **getCookies** on **HttpServletRequest** object
    - Returns an array of Cookie objects
    - Represents all cookies that server previously sent to the client
- For example, inside of **doPost**

```
Cookie[] cookies = request.getCookies();
```

Voiding Cookies

- May want to delete cookies when user logs out
  - Especially for sensitive information

```
// void cookie and send back to the user
userid_cookie.setMaxAge(0);
response.addCookie(userid_cookie);
```

Why Are They “Cookies”?

- Http Cookie, Source: Wikipedia
  - The term "cookie" derives from "magic cookie", which is a packet of data a program receives and sends out again unchanged.

- Magic Cookie, Source: Wikipedia
  - The name "cookie" comes from a comparison to an unopened fortune cookie, because of the hidden information inside.

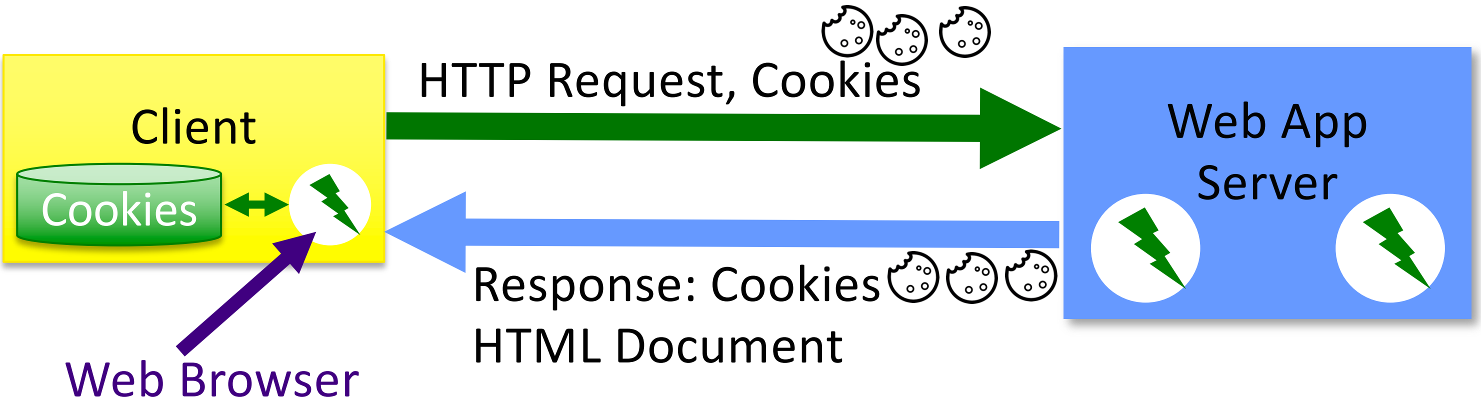
Look at the Cookies in Your Browser

- In Developer Tools
  - Chrome: Applications → Cookies
  - Firefox: Storage → Cookies

What are challenges with using cookies?

What are challenges with using cookies?

- They are saved on the client machine
  - Clients can delete or modify them



- Increase the sizes of your network packets
  - Send cookies on each request

Lab 5: Add Session Variable

- `LoginServlet` will add a session variable with name "authenticated"

TODO

- Lab 5: Servlet Configuration and Session State
  - Init, destroy methods
  - Configuration parameters
  - Session state
- Your web page – due tonight at 11:59 p.m.
- Read/Summarize Quality Attributes paper by Tuesday, 11:59 p.m.
  - See Canvas description for details about contents
- Tomorrow: JSPs, Introduce Projects